# Lab 1

THE REGISTER FILE AND ALU

DUE: 5 MARCH 2020

DAVID KIRBY
101652098

## Source Code

### RegisterFile.vhd

```
---------------------------------------------------
-- This source file describes a 32x32 register file such
-- that the 0th register is always 0 and is not writeable
---------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

---------------------------------------------------

entity RegisterFile is

port( RdRegA: in std_logic_vector(4 downto 0);
      RdRegB: in std_logic_vector(4 downto 0);
      WrReg: in std_logic_vector(4 downto 0);
      Clk: in std_logic;
      RegWrEn: in std_logic;
      WrData: in std_logic_vector(31 downto 0);
      RdDataA: out std_logic_vector(31 downto 0);
      RdDataB: out std_logic_vector(31 downto 0)
);
end RegisterFile;

---------------------------------------------------

architecture RegisterFile of RegisterFile is

---------------------------------------------------
--create an array of 31 32-bit registers
  type register_array is array (1 to 31) of
    std_logic_vector (31 downto 0);
  signal Registers: register_array;
---------------------------------------------------

begin

---------------------------------------------------
--describe the write functionality
  process(Clk)--only do something if clock changes
  begin
    --on the rising edge of clock
    if(Clk'event and Clk='1') then
      --only write if enabled and not
      --attempting to write to 0 reg
      if(RegWrEn='1' and conv_integer(WrReg)/=0) then
        Registers(conv_integer(WrReg)) <= WrData;
      end if;
    end if;
  end process;
```

```vhdl
---------------------------------------------------
--describe the read functionality
  process(RdRegA, Registers) begin
    if(conv_integer(RdRegA)=0) then RdDataA <=
      (others => '0'); --implements our $zero register
    else
      RdDataA <= Registers(conv_integer(RdRegA));
    end if;
  end process;

  process(RdRegB, Registers) begin
    if(conv_integer(RdRegB)=0) then RdDataB <=
      (others => '0'); --implements our $zero register
    else
      RdDataB <= Registers(conv_integer(RdRegB));
    end if;
  end process;
end RegisterFile;
---------------------------------------------------
```

## Source Code

### ALU.vhd

```vhdl
---------------------------------------------------
-- Arithmetic Logic Unit (ALU) takes at most two
-- 32-bit inputs and outputs one 32-bit result.
-- This is done via purely combinational logic.
---------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all; --needed to describe
use ieee.std_logic_arith.all; --arithmetic on std_logic_vector
use ieee.numeric_std.all; --types. Treated as unsigned!
entity ALU is

port( AluCtrl: in std_logic_vector(3 downto 0);
      AluInA, AluInB: in std_logic_vector(31 downto 0);
      AluResult: out std_logic_vector(31 downto 0)--;
--      Equals: out std_logic       -- not needed yet
);
end ALU;

architecture ALU of ALU is

begin

  process(AluInA, AluInB, AluCtrl)
  begin
    case AluCtrl is
              -- Bitwise ands two registers and stores the result in a register
          when b"0000" => --AND
              AluResult <= AluInA and AluInB;

              -- Bitwise logical ors two registers and stores the result in a
              -- register
          when b"0001" => --OR
              AluResult <= AluInA or AluInB;

              -- Shifts a register value left by the shift amount listed in the
              -- instruction and places the result in a third register. Zeroes
              -- are shifted in.
          when b"0011" => --SLL
              AluResult <= to_stdlogicvector(to_bitvector(AluInB) sll conv_integer(AluInA));

              -- Shifts a register value right by the shift amount (shamt) and
              -- places the value in the destination register. Zeroes are
              -- shifted in.
          when b"0100" => --SRL
              AluResult <= to_stdlogicvector(to_bitvector(AluInB) srl conv_integer(AluInA));

              -- Adds two registers and stores the result in a register
          when b"1000" => --ADDU
```

```
                AluResult <= AluInA + AluInB;

                -- Subtracts two registers and stores the result in a register
            when b"1001" => --SUBU
                AluResult <= AluInA - AluInB;

                -- Exclusive ors two registers and stores the result in a register
            when b"1010" => --XOR
                AluResult <= AluInA xor AluInB;

                -- If $s is less than $t, $d is set to one. It gets zero
                -- otherwise.
            when b"1011" => --SLTU
                if (AluInA) < (AluInB) then
                    AluResult <= (0 => '1', others => '0');
                else
                    AluResult <= (others => '0');
                end if;

                -- Nors two registers and stores the result in a register
            when b"1100" => --NOR
                AluResult <= AluInA nor AluInB;

                -- Shifts a register value right by the shift amount (shamt) and
                -- places the value in the destination register. The sign bit is
                -- shifted in.
            when b"1101" => --SRA
                AluResult <= to_stdlogicvector(to_bitvector(AluInB) sra conv_integer(AluInA));

                -- The immediate value is shifted left 16 bits and stored in the
                -- register. The lower 16 bits are zeroes.
            when b"1110" => --LUI
                AluResult <= AluInB(15 downto 0) & x"0000";

                -- Everything else
            when others => -- others
                AluResult <= (others => '-');
        end case;
    end process;
end ALU;
```